VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Petr Heinz

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Heinz**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: PDS s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura, Ph.D.**

Konzultant bakalářské práce: Ing. Jaroslav Košulič

Datum zadání: 01.09.2018
Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky * OSTRAVA *

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 1. dubna 2019

.....

PDS s.r.o.
Viniční 20
Brno, 615 00
IČ:25523121 DIČ:CZ25523121

Rád bych na tomto místě poděkoval své rodině za podporu při studiu, Ing. Petru Oujezdskému za nabídnutou možnost vykonat odbornou praxi ve firmě PDS s.r.o., Ing. Jaroslavu Košuličovi za vedení odborné praxe a poskytnuté odborné konzultace, Ing. Petru Doudovi, Ing. Ondřeji Svobodovi a Ing. Danielu Rychlému za poskytnuté odborné konzultace a Mgr. Bc. Janě Bezděkové Ph.D za jazykovou korekturu práce.

Abstrakt

Tato bakalářská práce popisuje odbornou praxi, kterou jsem absolvoval u firmy PDS s.r.o. po dobu 51 dní mezi zářím 2018 a březnem 2019. V práci je popsáno odborné zaměření firmy PDS s.r.o., mé pracovní zařazení a úkoly, které mi byly v průběhu praxe zadány, včetně popisu řešení těchto úkolů.

V druhé části práce popisuji, které znalosti a dovednosti nabyté v průběhu studia jsem uplatnil v průběhu své bakalářské praxe, které znalosti mi v průběhu praxe scházely a jakých výsledků jsem v průběhu praxe dosáhl, včetně celkového zhodnocení této praxe.

Klíčová slova: odborná praxe, Android, Xamarin, Xamarin.Forms, C#, .NET, .NET Standard, Cordova, JavaScript, JSON, OpenLayers, GeoJSON, GIS, mapa, odvozní lístek, lesnictví, cloud

Abstract

This bachelor thesis describes professional practice, which I was completing in the company PDS s.r.o for 51 days between September 2018 and March 2019. In the thesis, there is professional focus of the company PDS s.r.o, my position and tasks given to me duing the practice described, including descriptons of solutions to given tasks.

In the second part of the thesis, I cover, which knowledge and skills gained during my studies I used during my professional practice, which knowledge I lacked and which results I achieved during the practice, including overall summary of the practice.

Key Words: professional practice, Android, Xamarin, Xamarin.Forms, C#, .NET, .NET Standard, Cordova, JavaScript, JSON, OpenLayers, GeoJSON, GIS, map, delivery ticket, forestry, cloud

Obsah

Seznam použitých zkratek a symbolů

ADB	–	Android Debug Bridge
AGPL	–	Affero General Public License
AOT	–	Ahead Of Time (kompilace)
ART	–	Android Runtime
CIL	–	Common Intermediate Language
CLR	–	Common Language Runtime
CRS	–	Coordinate Reference System
CSS	–	Cascading Style Sheets
ČÚZK	–	Český úřad zeměměřičský a katastrální
DDL	–	Data Definition Language
DOM	–	Document Object Model
ER	–	Entity Relationships (diagram)
GPS	–	Globální poziční systém
GUI	–	Grafické uživatelské rozhraní
HTML	–	Hypertext Markup Languge
IIS	–	Internet Information Services
IMEI	–	International Mobile Equipment Identity
JAR	–	Java Archive
JIT	–	Just In Time (kompilace)
JSON	–	JavaScript Object Notation
LGPL	–	GNU Lesser General Public License
MVVM	–	Model-View-ViewModel
ORM	–	Objektově-relační mapování
OS	–	Operační systém
PDF	–	Portable Document Format
S-JTSK	–	Systém jednotné trigonometrické sítě katastrální
SDK	–	Software Development Kit
SHA	–	Secure Hash Algorithm
SQL	–	Structured Query Language
URL	–	Uniform Resource Locator
UWP	–	Universal Windows Platform
WGS	–	World Geodetic System
WKT	–	Well Known Text
WMS	–	Web Map Service
XAML	–	Extensible Application Markup Language
XML	–	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je popsat odbornou praxi, kterou jsem absoloval u firmy PDS s.r.o. mezi zářím 2018 a březnem 2019, v celkové délce trvání 51 dní. Níže v práci popisuji odborné zaměření firmy, mé pracovní zařazení a úkoly, které mi byly v průběhu praxe zadány, včetně jejich řešení. U komplikovanějších a zajímavějších úkolů přidávám i podrobnější popis řešení, včetně ukázek zdrojového kódu. V druhé části práce se snažím nastínit, které znalosti a dovednosti získané v průběhu studia mi byly v průběhu praxe prospěšné, které znalosti mi naopak scházely a hodnotím praxi jako celek.

Pro absolvování odborné praxe jsem se rozhodl hned z několika důvodů - chtěl jsem si vyzkoušet práci na reálném projektu v korporátním prostředí, získat nové zkušenosti a seznámit se s novými technologiemi pro vývoj software. Zároveň jsem si uvědomoval důležitost praxe pro budoucí uplatnění na trhu práce. V průběhu své odborné praxe jsem postupně pracoval na třech vzájemně provázaných projektech - vytvářel jsem novou mobilní aplikaci PDS_OLi, upravoval funkcionalitu existující aplikace ProPlaMobile a rozšiřoval webovou službu ProPlaCloud.

2 Firma PDS s.r.o.

Bakalářskou praxi jsem absolvoval u firmy PDS s.r.o, sídlící v Brně, vzniklé v roce 1998. Níže popisuji odborné zaměření firmy a své pracovní zařazení.

2.1 Odborné zaměření firmy

Firma se po svém vzniku specializovala na vytváření aplikací a systémů pro lesnictví. Ve známost vešla zejména díky lesní hospodářské evidenci PDS_ProPla, mezi jejíž největší uživatele patří například Lesy České Republiky a.s., Vojenské lesy a statky ČR a Arcibiskupské lesy a statky Olomouc.

Postupem času se činnost firmy rozšířila také o tvorbu informačních systémů pro zemědělství, životní prostředí, státní správu i o další oblasti. Firma později dosáhla významných úspěchů na poli eGovernmentu, což v současnosti patří ke stěžejní náplni její činnosti.

2.2 Pracovní zařazení studenta

Součástí týmu PDS jsem se stal již v listopadu 2017. Po absolvování pohovoru mi byl zadán zkušební úkol - úprava stávající funkcionality aplikace ProPlaMobile, vydávané na operační systém Android. V průběhu plnění tohoto úkolu jsem se seznámil s programovacím jazykem JavaScript a technologií Apache Cordova, umožňující tvorbu multiplatformních mobilních aplikací právě v tomto jazyce. Po úspěšném splnění zadaného úkolu jsem dále pracoval na úpravách a rozšiřování aplikace ProPlaMobile.

V červnu 2018 mi byla nabídnuta možnost absolvovat ve firmě bakalářskou praxi, jejímž cílem mělo být vytvoření nové mobilní aplikace PDS_OLi. V rámci této praxe jsem se seznámil s kompletním vývojovým cyklem mobilní aplikace pro operační systém Android, od úvodního návrhu až po vydání aplikace na Google Play. Vývoj nové aplikace probíhal na plaformě Xamarin.Forms v programovacím jazyce C#. Na vývoji aplikace jsem pracoval v zásadě sám, zvolené postupy a technologie jsem konzultoval se svým konzultantem bakalářské práce Ing. Jaroslavem Košuličem.

Kromě vývoje aplikace PDS_OLi jsem se nadále věnoval i rozšiřování aplikace ProPlaMobile - konkrétně funkcionality kreslení uživatelských náčrtů do mapy. V rámci své bakalářské praxe jsem vyvinul interní rozšíření existujícího mapového frameworku OpenLayers o funkcionalitu táhla pro posunování nakreslených bodů, což výrazně usnadnilo tvorbu uživatelských náčrtů na mobilních zařízeních. Ta byla bez funkcionality táhla značně nepřesná, neboť prst uživatele zakrýval samotný upravovaný bod náčrtu. V závěru praxe jsem se ještě seznámil s implementací cloudové služby ProPlaCloud a provedl její rozšíření o služby související s výstupy z aplikace PDS_OLi.

3 Použité technologie

Při vykonávání praxe jsem se setkal s velkým množstvím technologií, frameworků a knihoven. V této kapitole popisuji významnější z těchto použitých technologií.

3.1 Android

Android je mobilní operační systém používaný zejména na smartphonech, tabletech, chytrých televizích a ostatních zařízeních. Je založen na jádře Linuxu a vývoj je veden firmou Google. První verze tohoto operačního systému vyšla v roce 2008 a v současnosti se jedná o nejrozšířenější operační systém pro mobilní zařízení. Vývoj nativních aplikací pro operační systém Android probíhá v programovacím jazyce Java.

3.1.1 ADB

ADB - Android Debug Bridge je nástrojem, který umožňuje komunikaci s zařízením prostřednictvím příkazové řádky. Je programem fungujícím na principu klient-server a skládá se ze tří komponent - klienta, démona a serveru.

3.1.2 APK

Je formát pro distribuci aplikací pro operační systém Android. Soubory APK jsou ZIP archivy na bázi formátu JAR - Java Archive.

3.1.3 SDK

Android SDK - Software Development Kit je balíčkem nástrojů určených pro vývoj aplikací pro platformu Android. Obsahuje například emulátor, debugger, potřebné knihovny, ale i tutoriály a ukázky zdrojových kódů.

3.1.4 ART

ART - Android Runtime je název virtuálního stroje používáného pro běh aplikací napsaných v programovacím jazyce Java na platformě Android. Od Androidu verze 5.0 nahradil původní virtuální stroj - Dalvik.

Oproti Dalviku umožnuje *AOT - Ahead of Time* kompilaci zdrojového kódu - to znamená, že se při instalaci aplikace zkompiluje do strojového kódu místo užití *JIT - Just in Time kompilace*, při které se kód kompiloval do strojového kódu až ve chvíli, kdy to bylo potřeba. To přineslo výrazné zrychlení běhu aplikací i zvýšení výdrže baterie.

3.2 Apache Cordova

Je framework pro vývoj hybridních multiplatformních mobilních aplikací s použitím značkovacího jazyka HTML, kaskádových stylů CSS a programovacího jazyka JavaScript. Takto vytvořené webové aplikace mohou přistupovat k nativním funkcionalitám operačních systémů pomocí pluginů. Mezi podporované platformy patří kromě Androidu také iOS, BlackBerry, Bada, Firefox OS, webOS a další.

3.3 Microsoft .NET

Je vývojářskou platoformou. Mezi její implementace patří například .NET Framework, .NET Core nebo Xamarin (Mono). Umožňuje vývoj aplikací v programovacích jazycích C#, Visual Basic .NET, F# a dalších - zdrojový kód napsaný v jakémkoliv z těchto programovacích jazyků je kompilován do kódu CIL - Common Intermediate Language, což umožňuje mít kteroukoliv část programu napsanou v kterémkoliv z těchto jazyků. Kód CIL je spouštěn pomocí virtuálního stroje CLR - Common Language Runtime.

3.3.1 C#

C Sharp je vysokoúrovňový objektově orientovaný programovací jazyk založený na jazycích C++ a Java, se kterými má velmi podobnou syntaxi, vyvinutý firmou Microsoft společně s platformou .NET.

3.3.2 .NET Standard

Je způsobem, jak alespoň částečně sjednotit jednotlivé implementace .NETu - obsahuje tedy ty knihovny, které jsou pro všechny implementace .NETu společné. Balíčky sestavené pro .NET Standard tak jsou kompatibilní s kteroukoliv z těchto implementací .NETu. Příkladem může být hypotetická knihovna pro výpočet obsahu čtverce, kterou lze bez úprav použít v mobilní aplikaci s použitím Xamarin.Androidu i v desktopové aplikaci běžící pod .NET Frameworkem.

3.4 SQLite

Je relační databázový systém. Narozdíl od databázových systémů založených na princpu klient-server je SQLite databáze pouze malým souborem uloženým na disku. Pro dotazování nad databází se používá dotazovací jazyk SQL.

3.4.1 Mbtiles

Je formátem pro ukládání mapových dlaždic. Technicky se jedná o SQLite databázi, která kromě binárních dat reprezentujících jednotlivé dlažice obsahuje ještě metadata určující např. formát, rozsah a hranice mapových dlaždic a minimální/maximální úroveň přiblížení.

3.5 WMS

WMS - Web Map Service, v překladu webová mapová služba je služba pracující na principu klient-server. Umožňuje sdílení a distribuci geografických informací ve formě mapových dlaždic.

3.6 Xamarin.Forms

Je technologií umožňující vývoj nativních multiplaformních aplikací pro operační systémy Android, iOS a Windows Phone (UWP - univerzální Windows aplikace) pomocí značkovacího jazyka XAML a programovacího jazyka C#. Takto vytvořená aplikace přizpůsobí svůj vzhled nativním ovládacím prvkům a umožní sdílení podstatného množství kódu napříč platformami. Xamarin.Forms je kompatibilní s technologií .NET Standard, což umožňuje užití množství balíčků dostupných skrz správce balíčků *NuGet*.

3.6.1 Model-View-ViewModel

Je návrhový vzor oddělující logiku aplikace od uživatelského rozhraní. Skládá se ze tří komponent - Modelu, reprezentujícího datový model aplikace, View, které reprezentuje grafické uživatelské rozhraní a ViewModelu, který zastřešuje stav aplikace. View je s ViewModelem spojeno na základě tzv. *Data Bindingu* - svázání datových zdrojů. Ten může být jednosměrný, nebo obousměrný. Podle toho může View aktualizovat ViewModel (např. změna vybraného objektu v ComboBoxu), nebo ViewModel aktualizovat View (např. změna svázané vlastnosti se promítne změnou vybraného objektu v ComboBoxu). ViewModel musí implementovat rozhraní *INotifyPropertyChanged*. Model a View spolu komunikují pouze skrz ViewModel. Příkladem užití tohoto vzoru je prezentační framework WPF.

3.6.2 XAML

XAML - Extensible Application Markup Language je značkovací jazyk určený pro popis grafického uživatelského rozhraní použitý např. v technologiích WPF - Windows Presentation Framework a Xamarin.Forms. Je založen na XML.

4 Zadané úkoly

Cílem této kapitoly je nastínit, jaké úkoly a podúkoly mi byly v průběhu odborné praxe zadány. Dále uvádím, jaká byla časová náročnost zadaných úkolů.

4.1 Tvorba aplikace PDS_OLi

Prvním ze zadaných úkolů bylo vytvoření nové multiplatformní mobilní aplikace PDS_OLi pomocí frameworku Xamarin.Forms, v programovacím jazyce C#. V úvodu jsem měl vytvořit obyčejnou formulářovou aplikaci, umožňující správu odvozních lístků. To zahrnovalo níže uvedené podúkoly:

- Seznámení se s platformou Xamarin a Xamarin.Forms
- Tvorba grafického uživatelského rozhraní
- Analýza dostupných ORM frameworků
- Tvorba datového modelu a implementace datové vrstvy
- Seznámení se s návrhovým vzorem MVVM a jeho implementace v aplikaci
- Implementace mechanismu pro aktualizaci databáze
- Vytvoření release APK

Další podúkoly pak zahrnovaly rozšiřování této formulářové aplikace o další funkcionalitu, zejména o export odvozních lístků - ať už do dokumentu aplikace *Microsoft Excel* nebo do aplikace PDS_ProPla pomocí webové služby ProPlaCloud a zobrazování odvozních míst uložených odvozních lístků v mapě. Na závěr byla beta verze aplikace vydána v obchodě Google Play. Časová náročnost úkolu byla 38 pracovních dnů.

4.2 Zjednoduššení tvorby uživatelských náčrtů v aplikaci ProPlaMobile

Druhým zadaným úkolem bylo zjednoduššení tvorby uživateslkých náčrtů v aplikaci ProPlaMobile. Cílem bylo implementovat táhlo pro editaci těchto náčrtů na dotykovém displeji mobilního telefonu a umožnit přichytávání náčrtů k již existujícím náčrtům a dalším vektorovým prvkům v mapě. Časová náročnost úkolu byla 12 pracovních dnů.

4.3 Rozšíření webové služby ProPlaCloud

Posledním zadaným úkolem bylo vhodně rozšířit firemní webovou službu ProPlaCloud o nahrávání a stahování odvozních lístků z aplikace PDS_OLi vyvinuté v rámci prvního úkolu. Časová náročnost úkolu byla 1 pracovní den.

5 Řešení zadaných úkolů

V této kapitole popisuji řešení úkolů, které mi byly v průběhu odborné praxe zadány. U komplikovanějších podúkolů přikládám podrobný popis jejich řešení, včetně ukázek zdrojových kódů či diagramů.

5.1 Tvorba aplikace PDS_OLi

Cílem aplikace PDS_OLi je nahradit v současné době používané papírové odvozní lístky za digitální. Odvozní lístek je doklad o množství odvezeného dříví dle dřevin, dodavatele, odběratele a ostatních požadovaných parametrů.

Uživatelé aplikace získají možnost vytvořit digitální odvozní lístek, vyplnit jej na mobilním zařízení - mobilním telefonu nebo tabletu přímo v lese, připojit fotografie odvozní soupravy a odeslat jej ihned lesnímu hospodáři k automatickému zaevidování v lesní hospodářské evidenci PDS_ProPla. Zákazníkovi pak umí aplikace vystavit elektronický doklad ve formátu XLSX - sešit Microsoft Excel.

Digitální odvozní lístek je na pevno svázán s GPS pozicí odvozu. Uživatel může prohlížet a vyhledávat uložené odvozní lístky buď na základě filtrace podle určených kritérií, nebo přímo v mapě na základě pozice odvozu. Mapové okno umožňuje rovněž prohlížení lesnických map exportovaných pomocí firemních aplikací KonverzeIslh a PDS_KoPla pro usnadnění orientace uživatele.

5.1.1 Seznámení se s platformou Xamarin a Xamarin.Forms

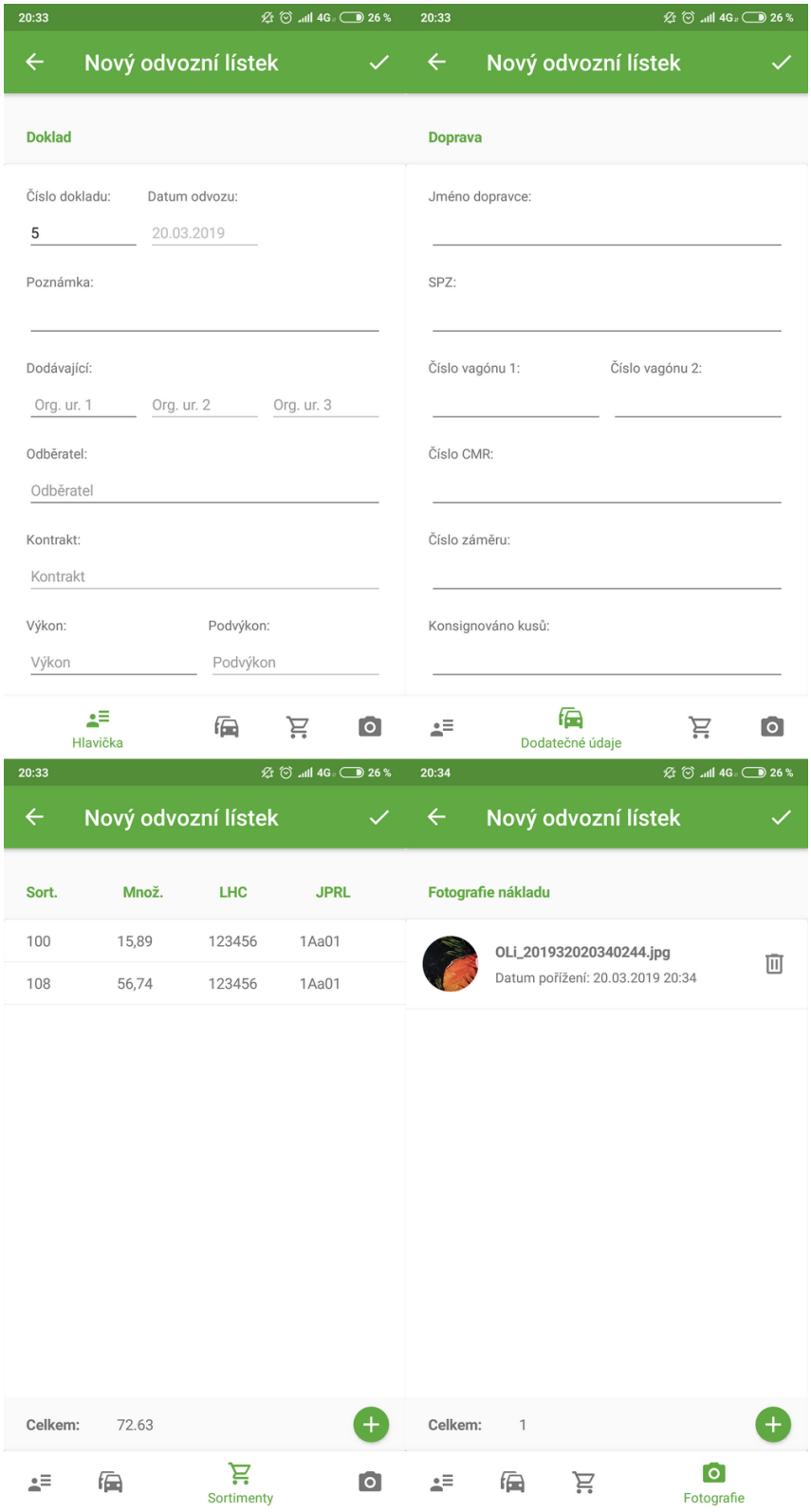
Prvním úkolem bylo seznámení se s navrhovanou platformou Xamarin a její multiplatformní nadstavbou Xamarin.Forms. V rámci tohoto úkolu jsem nainstaloval potřebné balíčky a vyzkoušel si spustit vygenerovaný projekt na svém mobilním zařízení.

Poté jsem postupným zkoumáním kódu pochopil hiearchii Xamarin.Forms projektu. Dále jsem se seznámil se značkovacím jazykem XAML určeným pro definici uživatelského rozhraní Xamarin.Forms aplikací a službou pro stahování balíčků NuGet. Na závěr jsem si vyzkoušel vytvořit jednoduchý přihlašovací formulář.

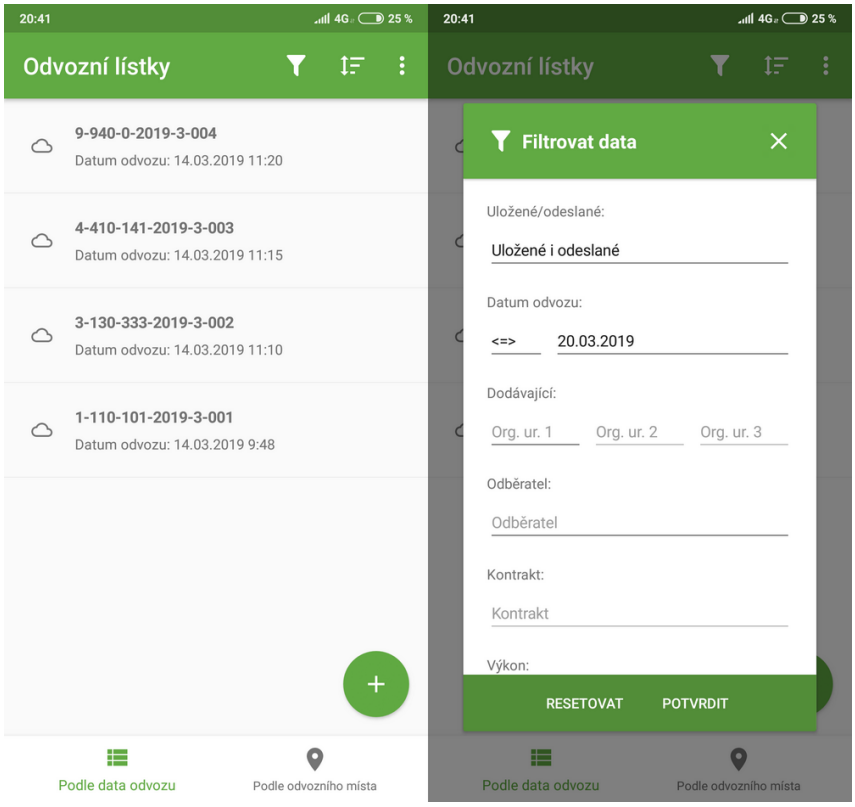
5.1.2 Tvorba grafického uživatelského rozhraní

Po základním seznámení se s platformou jsem ihned mohl začít vytvářet uživatelské rozhraní aplikace pomocí značkovacího jazyka XAML. Výsledek své práce jsem konzultoval s kolegy a případně jsem provedl úpravy vzhledu na základě jejich zpětné vazby.

Postupně jsem se seznámil i s pokročilejšími aspekty značkovacího jazyka XAML - například s užitím stylů. Pro zjednoduššení zápisu jsem tak na závěr tvorby GUI provedl úpravy kódu s cílem zapsat co nejvíc vlastností vzhledu pomocí stylů - aby stačilo kdykoliv upravit pouze styl a nebylo třeba měnit vlastnosti každého prvku zvlášť.



Obrázek 1: Uživatelské rozhraní přidání nového odvozního lístku



Obrázek 2: Uživatelské rozhraní seznamu odvozních lístků

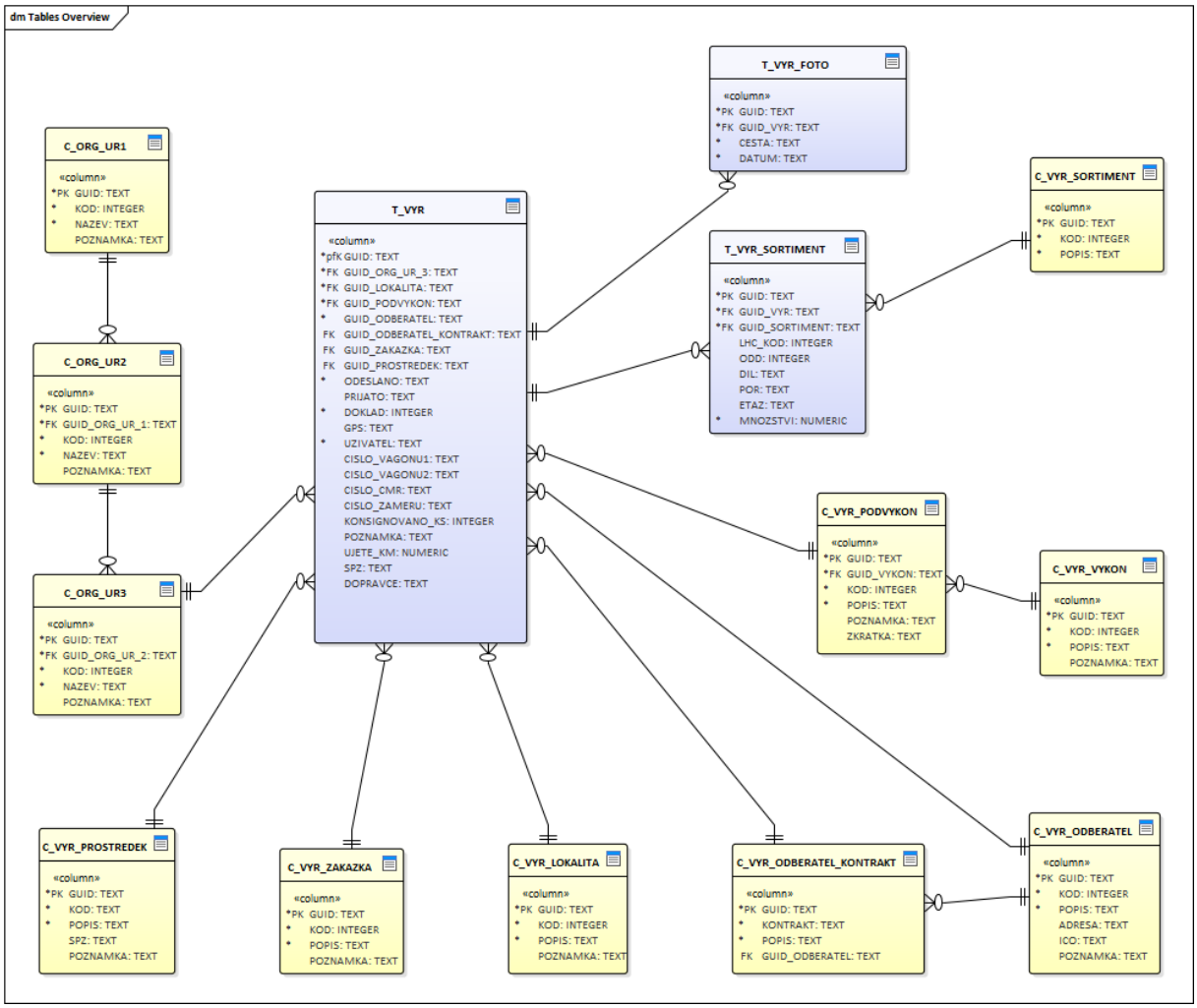
5.1.3 Analýza dostupných ORM frameworků

Cílem analýzy bylo najít vhodný framework pro ORM - objektově-relační mapování, tedy mapování mezi objekty a řádky v tabulkách relačních databází, který by fungoval s Xamarinem. I přesto, že se ve většině starších firemních projektů používal framework *nHibernate*, padla bez delších debat volba na *Entity Framework Core* - měl podporu pro Xamarin.Forms, kvalitní dokumentaci a bylo snadné ho použít. Pro užití *Entity Frameworku* bylo nutné provést přechod ze sdíleného projektu, který dosud zaštiťoval celou Xamarin.Forms aplikaci, na technologii *.NET Standard*.

5.1.4 Tvorba datového modelu a implementace datové vrstvy

Datový model vznikl zjednodušením datového modelu pro odvozní lístky z aplikace PDS_ProPla. V aplikaci *Enterprise Architect* jsem vytvořil ER diagram, na základě kterého aplikace vygenerovala DDL příkazy. Tyto příkazy pak posloužily pro vytvoření *SQLite* databáze.

Entity Framework nabízí možnost reverse-engineeringu datového modelu - tedy vygenerování příslušných tříd na základě databáze. K tomu slouží příkaz *Scaffold-DbContext* zadávaný do konzole správce balíčků - ten kromě tříd reprezentujících datový model vygeneruje i speciální třídu *DbContext*, skrz kterou probíhá veškerý přístup k databázi.



Obrázek 3: Datový model aplikace PDS_OLI

```
using (var db = new DbContext())
{
  db.Lokalita.Add(new Lokalita
  {
    Kod = 111
  });
  db.SaveChanges();
}
```

Výpis 1: Přístup k databázi skrz DbContext a Entity Framework

5.1.5 Seznámení se s návrhovým vzorem MVVM a jeho implementace v aplikaci

S vytvořeným uživatelských rozhraním a datovým modelem bylo ještě nutné obě tyto složky propojit, aby vznikla použitelná aplikace. Xamarin.Forms využívá, podobně jako třeba WPF,

návrhového vzoru MVVM - Model-View-ViewModel. Bylo tedy nutné se s tímto návrhovým vzorem seznámit a vytvořit příslušné ViewModely - ke každému View právě jeden.

Z pohledu vývoje se v tomto případě jednalo o rutinní záležitost, která je sice časově náročná, ovšem nijak zajímavá ani složitá. Výsledkem byla standardní formulářová aplikace - vyplnění formuláře, jeho uložení do databáze a poskytnutí už uložených formulářů uživateli. Zajímavější byla pouze implementace fotografování odvozních souprav - díky použití stažitelného balíčku místo nativní implementace však značně zjednodušená.

5.1.6 Implementace mechanismu pro aktualizaci databáze

Protože může v budoucnu docházet k rozšiřování aplikace a není zároveň možné vždy při nutnosti aktualizace databáze zcela vyměnit jednu *SQLite* databázi za jinou - neboť by uživatelé ztratili uložená data, bylo potřeba implementovat mechanismus pro aktualizaci databáze.

Firma již krátce předtím zpracovala návrh jednotného mechanismu pro aktualizaci databáze, který hodlá v budoucnu implementovat ve všech svých projektech. Tento mechanismus spočívá v tvorbě tzv. *Changelog XML* souborů a jejich následném zpracování aplikací a provedení aktualizací.

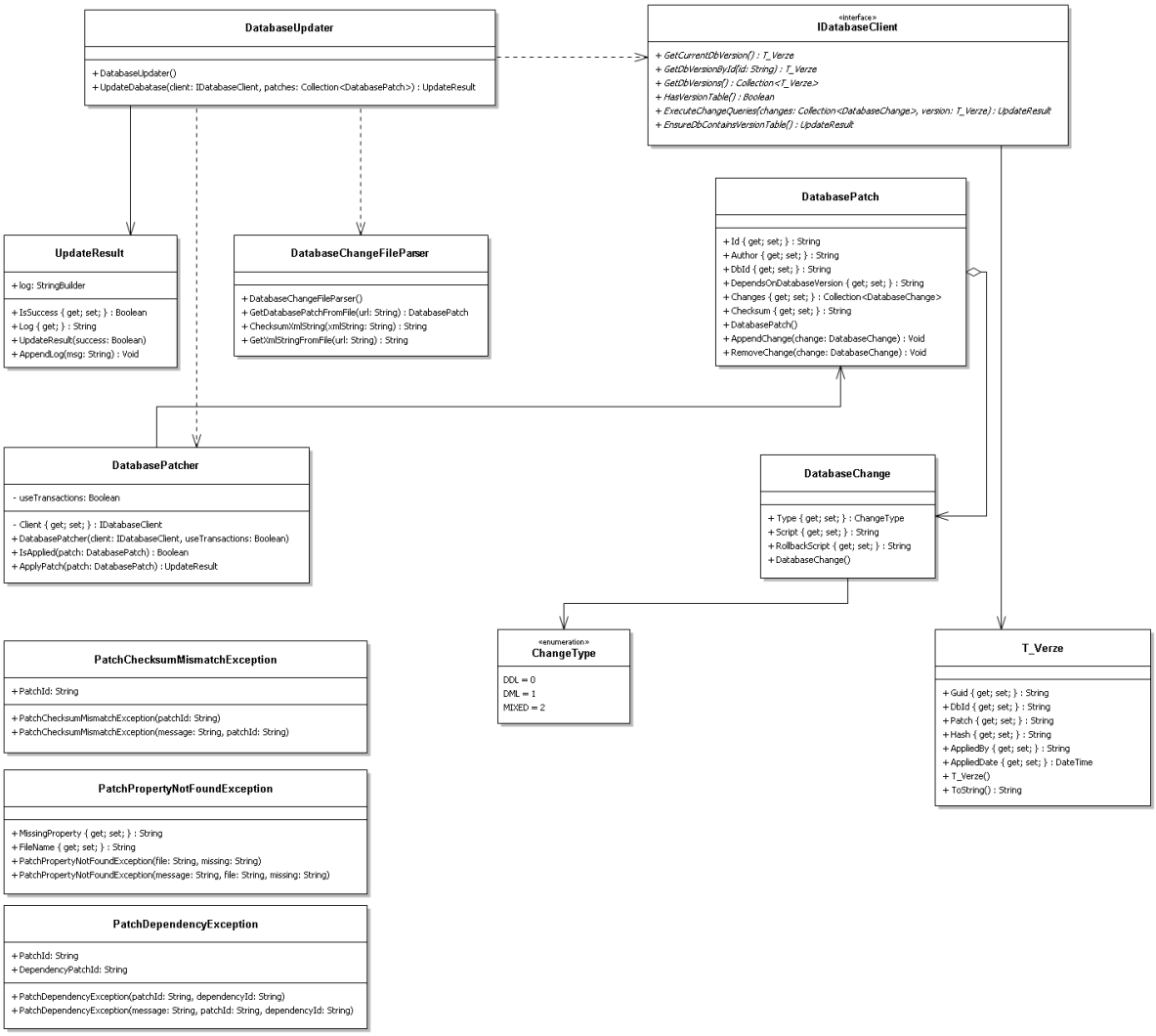
```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <DbPatch id="2019-02-01_1">
3    <Author>pheinz</Author>
4    <Created>2019-02-01</Created>
5    <Ticket>#17451</Ticket>
6    <DependsOnDbPatch>2018-10-18_1</DependsOnDbPatch>
7    <Change Type="ddl">
8      <Script>
9        <![CDATA[ALTER TABLE T_VYR ADD IDENTIFIKATOR TEXT;]]>
10     </Script>
11   </Change>
12 </DbPatch>
```

Výpis 2: Příklad Changelog XML

Výsledné řešení jde rozdělit do dvou částí - sdílený kód, který zaštiťuje logiku aktualizací a nativní implementaci přístupu k databázi implementující rozhraní *IDatabaseClient*.

Celou aktualizaci zaštiťuje třída *DatabaseUpdater*, jejíž instanci se předá databázový klient implementující rozhraní *IDatabaseClient* a seznam adres souborů se změnami (Changelog XML souborů). Poté jsou načteny jednotlivé soubory se změnami, zparsovány a vytvořeny příslušné instance třídy *DatabasePatch*.

Pokud ještě nebyly v minulosti provedeny načtené aktualizace, databázový klient spustí příslušné skripty. Pokud už byly, pouze se provede kontrola *SHA256* hashe změnového souboru - jestli nebyl tento soubor po aktualizaci upravován.



Obrázek 4: Třídní diagram mechanismu pro aktualizaci databáze v aplikaci PDS_OLi

Pokud se hashe neshodují, je vyhozena výjimka *PatchChecksumMismatchException* a aktualizace přerušena. Aktualizace databáze probíhá formou transakcí - pokud se nezdaří spuštění jednoho ze změnových skriptů, provede se *rollback*. Jeden *DatabasePatch* - jedna transakce. Tak je zajištěno, že je databáze vždy v korektním stavu.

5.1.6.1 Unit testování mechanismu pro aktualizaci databáze

Po dokončení implementace výše zmíněného mechanismu jsem byl požádán a vytvoření unit testů. Pro testovaní jsem využil balíčku *xunit*. Pro účely testování jsem vytvořil třídu *MockDatabaseClient* implementující rozhraní *IDatabaseClient* - abych se vyhnul závislosti výsledku testů na externí databázi. Při testování jsem nenarazil na žádné chyby v původní implementaci.

5.1.7 Vydání aplikačního balíčku

Výsledkem předchozích úkolů byl použitelný prototyp aplikace, který mohl být předán k testování. Byl jsem tedy požádán o vydání aplikačního balíčku. Po změně konfigurace sestavení z *Debug* na *Release* nebyla aplikace kvůli výjimce při startu spustitelná. Řešením problému byla změna *Release* konfigurace sestavení - povolil jsem *Proguard* a změnil vlastnosti linkeru (propojovače) tak, aby se sestavení nepropojovala. Poté již nebyl s funkcí aplikace žádný další problém.

Dále bylo nutné aplikaci podepsat. Požádal jsem kolegu o vygenerování nového úložiště klíčů, které jsem připojil v nastavení sestavení.

Posledním problémem byla velikost aplikace - více než 60 MB se mi zdálo u malé aplikace, jako je PDS_OLi příliš. Vzhledem k použití technologie Xamarin.Forms se s tím však nedalo nic víc udělat. Existuje možnost použití sdíleného Mono runtime modulu, pro release verze aplikace to však nelze doporučit - pokud není Mono runtime modul v zařízení nainstalován, aplikaci by nešlo spustit. U debug sestavení však umí použití sdíleného runtime modulu značně snížit velikost aplikace - v případě PDS_OLi z 60 MB na pouhých 18 MB.

5.1.8 Optimalizace aplikace

Vážným problémem vytvořené aplikace byl její pomalý běh. Cílem tohoto úkolu tak bylo provést co nejvíce optimalizací a alespoň částečně přiblížit rychlost běhu Xamarin.Forms aplikace nativním aplikacím psaným v jazyce Java.

Největší negativní vliv na výkon měla ORM knihovna Entity Framework Core - první volání způsobilo zásek aplikace až na několik sekund. Bylo tedy jasné, že bude nutné ji nahradit za některý z microORM frameworků, nebo si napsat vlastní objektově relační mapování.

Dále jsme se rozhodli pro některé optimalizace uživatelského rozhraní - zejména pro lazy-loading podstránek v *TabbedPage*. Všechny podstránky se totiž načítaly najednou, což mohlo při větším počtu prvků značně zpomalit běh aplikace. Posledním nápadem, jak zrychlit běh aplikace bylo použití dopředné - AOT kompilace místo standardně používané JIT kompilace.

5.1.8.1 Nahrazení Entity Frameworku za Dapper

Po provedení analýzy dostupných microORM frameworků padla volba na Dapper - firma už jej používala v některém z předchozích projektů. Přes veškerou snahu se vytvořeným rozhraním co nejvíce přiblížit k tomu, co poskytoval Entity Framework se svým DbContextem, byl tento úkol značně časově náročný - vyžadoval totiž zásah do takřka všech částí projektu.

U Dapperu bylo narozdíl od Entity Frameworku nutné napsat si vlastní třídy pro přístup k databázi a také vlastní SQL dotazy. Pro rozhraní datové vrstvy jsem využil návrhové vzory Unit of Work v kombinaci s vzorem Repository, což ve výsledku ještě zjednodušilo celý přístup k databázi z ViewModelů - například samotný odvozní lístek se díky vzoru Repository tvářil jako jednotný celek, i když byl v databázi reprezentován třemi tabulkami - jednou pro hlavičku

lístku, jednou pro jeho sortimenty a jednou pro jeho fotografie. Odstranění Entity Frameworku mělo výrazný vliv na zlepšení výkonu aplikace, délka startu aplikace klesla až o šest sekund.

5.1.8.2 Lazy-loading podstránek v TabbedPage

Pří průzkumu možností na internetových fórech zabývajících se problematikou tvorby aplikací pro Xamarin.Forms jsem narazil na Github repozitář *TabbedPageLazyLoadApp*. Použitím zde implementovaného mechanismu jsem dosáhl lazy-loadingu podstránek. Marginální vliv na výkon to však na mém zařízení nemělo - může to však mít vliv na výkon na slabších mobilních telefonech.

5.1.8.3 Animace přechodů mezi obrazovkami

Pro vytvoření zdání plynulosti jsem se rozhodl vykoušet přidání animací přechodů mezi obrazovkami. Pro tento účel jsem použíl balíček *XForms.Plugin.AnimationNavigationPage* a odladil animace tak, aby jejich délka přibližně odpovídala době nutné pro načtení nové obrazovky.

5.1.8.4 AOT kompilace

Povolení AOT kompilace v nastavení sestavení mělo výrazný vliv na výkon aplikace - délka startu aplikace klesla až na pouhé 4 sekundy, před všemi optimalizacemi trvala leckdy i 12 sekund. Výrazně se zlepšil i běh uživatelského rozhraní - přechody mezi obrazovkami byly náhle plynulejší.

5.1.9 Implementace exportu odvozního lístku do PDF

Je potřeba, aby šlo vytvořený odvozní lístek z aplikace exportovat do formátu vhodného pro tisk. Z tohoto hlediska se jako nejvhodnější formát jevilo *PDF*. V úvodu jsem provedl analýzu dostupných knihoven pro tvorbu PDF v jazyce C# a nalezl několik bezplatných variant. Rozhodli jsme se vyzkoušet dvě možnosti - PdfSharp a iTextSharp. Tyto varianty jsem se postupně pokusil rozběhnout v prostředí Xamarin.Androidu.

Bohužel se mi ani jeden z dostupných balíčků nepovedlo uvést do chodu - důvodem byla buď závislost na knihovně *System.Drawing.Common*, která není dostupná pro *Mono.Android*, nebo neexistující *Mono.Android* profil balíčku.

Nejnovější verzi balíčku iTextSharp jsme nemohli použít z důvodu přechodu z LGPL na AGPL licenci. Z těchto důvodů jsme se s kolegy dohodli, že export do PDF nebudeme prozatím řešit a na místo toho nabídneme uživatelům export do formátu XLSX - tedy do souborů aplikace *Microsoft Excel*.

Tabulka 1: Testované knihovny pro export do PDF a vzniklé problémy

Knihovna	Problém
PdfSharp a MigraDoc	PdfSharp.dll neexistuje v profilu Mono.Android
PdfSharp.nestandard	Nelze nalézt assembly System.Drawing.Common
iTextSharp	AGPL licence
iTextSharp-LGPL-Core	Nelze nalézt assembly System.Drawing.Common

5.1.10 Implementace mapové složky

Nejspíš nejrozsáhlejším a zároveň nejzajímavějším úkolem byla implementance mapového okna podobného tomu v aplikaci ProPlaMobile - zobrazení podkladových map od ČÚZK, lesnických map generovaných pomocí firemní aplikace PDS_KoPla ve formátu *Mbtiles* a zobrazení odvozních lístků podle místa odvozu - GPS pozice jejich uložení.

Problémů bylo přitom hned několik - v aplikaci ProPlaMobile se pro mapové okno používá JavaScriptová knihovna *OpenLayers*, která svým rozsahem a intuitivností nemá v .NET univerzu rozumnou alternativu - bylo tak nutné zvolit jeden z méně rozvinutých balíčků a chybějící funkce doimplementovat.

Dalším z problémů byl formát Mbtiles - pro použití v aplikaci ProPlaMobile jsou Mbtiles mapy generované v aplikaci PDS_KoPla odlišné od specifikace - dlaždice jsou ukládny ve formátu *Base64* místo binární reprezentace, definice mapy pak je v samostatném XML souboru místo metadat přímo v Mbtiles databázi.

Situaci ztížilo i použití souřadnicového systému (CRS/SRS) *EPSG:102067* v generovaných lesnických mapách - to znamenalo, že bude nutná transformace standardně používaných *WGS-84* souřadnic (souřadnice GPS) do formátu *S-JTSK Krovak East North*.

Posledním problémem byl přímo Xamarin - bylo nutné, aby použitá knihovna měla *Mono.Android* profil, nebyla závislá na knihovně *System.Drawing.Common*, se kterou jsme měli problémy už při implementaci exportu odvozního lístku do PDF a aby existovala alespoň jedna knihovna pro geodetické transformace, která by fungovala na Xamarinu.

Bylo mi zadáno nejprve vyzkoušet balíček *Mapsui*, jelikož s ním už kolegové pracovali v projektu PDS_KoPla. Nainstaloval jsem balíček ve verzi *beta18*, jelikož tato verze již narozdíl od stabilní verze obsahovala podporu pro Xamarin.Forms. Stabilní verze byla dostačující pro běh balíčku v prostředí Xamarinu, bylo by však nutné implementovat mapovou funkcionalitu nativně a zvlášť pro každou platformu.

5.1.10.1 Zobrazení mapového okna

Prvním z podúkolů bylo zobrazení samotného mapového okna. Mapové okno je v balíčku *Mapsui.Forms* implementováno jako XAML prvek, stačilo tedy přidat jej do nově vytvořeného *MapView*. Toto mapové okno samozřejmě ještě neobsahovalo mapu.

Ovládací prvky nebyly dle mého názoru graficky zdařilé a rozhodl jsem se je nepoužít - až na měřítko. Tlačítka pro přiblížení/oddálení se v době dotykových gest dají považovat za zbytečná - z toho důvodu jsem implementoval pouze nové tlačítko pro načtení aktuální GPS pozice v souladu s designovým jazykem aplikace a tlačítko pro přidání nového odvozního lístku. V toolbaru mapového okna jsem se rozhodl pro zobrazení tlačítek pro výběr zobrazených mapových vrstev a pro přesun nad lesnickou mapu.

5.1.10.2 Načítání WMS vrstev

Jako podkladovou vrstvu jsme se rozhodli nabídnout uživatelům buď ortofoto, nebo základní mapu od ČÚZK. Obě tyto mapy jsou distribuovány skrz WMS server. Knihovna *Mapsui* nabízí příslušné třídy a rozhraní pro snadné zprovoznění stahování mapových dlaždic z WMS serverů a způsob, jak toho dosáhnout ukazuje v balíčku *Mapsui.Samples*, který obsahuje příklady použití nejběžnějších funkcí knihovny. V mém případě tak bylo nutné doladit zejména finální podobu HTTP požadavku, aby nebyl serverem zamítnut s chybou. Takto vytvořený požadavek pak stačí obalit do třídy reprezentují zdroj dlaždicových dat - tedy jakékoliv třídy implementující rozhraní *ITileSource* a vytvořit novou mapovou vrstvu.

```
1  private static WmscRequest CreateWmsRequest(ITileSchema schema)
2  {
3    const string url = "http://geoportal.cuzk.cz/WMS_ORTOFOTO_PUB/WMService.aspx?";
4    Dictionary<string,string> param = new Dictionary<string,string> ();
5    param.Add("CRS", "EPSG:102067");
6    var layers = new[] {"GR_ORTFOTORGB"}.ToList();
7    var styles = new[] {"default"}.ToList();
8    var version = "1.3.0";
9    return new WmscRequest(new Uri(url), schema, layers, styles, param, version);
10 }
```

Výpis 3: Ukázkový WMS požadavek

```
1  public CuzkOrtoWmsTileSource()
2  {
3    var schema = new GlobalSphericalMercator()
4    {
5      Srs = "EPSG:102067",
6      Format = "image/png"
7    };
8    Provider = new HttpTileProvider(CreateWmsRequest(schema));
9    Schema = schema;
10 }
```

Výpis 4: Konstruktor zdroje dlaždicových dat

```
private TileLayer _cuzkOrtofoto;
public TileLayer CuzkOrtofoto
{
  get
  {
    if (_cuzkOrtofoto == null) _cuzkOrtofoto = new TileLayer(new CuzkOrtoWmsTileSource())
    {
      CRS = "EPSG:102067"
    };
    return _cuzkOrtofoto;
  }
}
```

Výpis 5: Příklad lazy-loaded mapové vrstvy

5.1.10.3 Načítání Mbtiles vrstev

Druhým podúkolem bylo načítat lesnické vrstvy generované v aplikaci PDS_KoPla. Podobně jako u WMS vrstev, i načítání Mbtiles vrstev bylo ukázáno na příkladu v balíčku *Mapsui.Samples*. Zde však, jak už jsem nastínil v úvodu, byla celá situace komplikovanější.

Rozhodl jsem se převzít existující třídu pro načítání Mbtiles vrstev, implementovanou přímo v rámci balíčku *Mapsui* a upravit si ji pro potřeby aplikace. Původně jsem měl v plánu pouze vytvoření potomka této třídy a přepsání metod, ale vzhledem k tomu, že metody v původní třídy nebyly označené jako virtuální, nebylo by takové překrytí možné.

V převzaté třídě jsem předělal načítání metadat z databáze na načtání metadat z XML souboru. Tyto metadata například specifikují, v jakém rozlišení a na jaké pozici v souřadnicovém systému se mají data zobrazit. Po jejich úspěšném zpracování bylo možné korektně zobrazit načtené Mbtiles dlaždice v aplikaci.

Dále bylo nutné provést změny u zpracování načtených dat dlaždic. Dle specifikace Mbtiles by se měla z databáze načítat data v binární reprezentaci, ve výstupu z aplikace PDS_KoPla však byly zakódovány jako *Base64*. Bylo tedy nutné provést konverzi dat na binární reprezentaci.

5.1.10.4 Zobrazení aktuální GPS pozice a transformace souřadnic

Pro zjištění GPS pozice lze použít funkcionality, kterou nabízí balíček *Xamarin.Essentials*. Před zjištěním pozice je nutné vyžádat si od uživatele oprávnění. Toto jsem již měl vyřešeno v rámci přidávání nových odvozních lístků a tak stačilo zavolat příslušnou metodu pro zjištění GPS pozice.

Zjištěná pozice je však ve formátu *WGS-84* a je nutné ji transformovat do souřadnicového systému užívaného mapou - v našem případě do *EPSG:102067*. Pro tento účel jsem využil balíček *ProjNet4GeoAPI*, který jako jediný ze zdarma dostupných balíčků fungoval i s Xamarinem.

Pro definici souřadnicového systému slouží WKT, na základě kterého je poté transformační knihovna schopná provést převod z jednoho souřadnicového systému do druhého. Pro mé potřeby dostačilo zjištění WKT pro *EPSG:102067*, neboť *WGS-84* WKT již knihovna znala. Vytvořil jsem transformační třídu pro transforamce do *S-JTSK Krovak East North*, kterou používám napříč celým projektem - například i pro transformaci měřítka. Po transformaci souřadnic už stačilo pouze vytvořit novou prázdnou mapovou vrstvu a přidat do ní nový geografický prvek (featuru) - bod.

```
PROJCS["S-JTSK_Krovak_East_North",
    GEOGCS["GCS_S_JTSK",
        DATUM["Jednotne_Trigonometricke_Site_Katastralni",
            SPHEROID["Bessel_1841",6377397.155,299.1528128]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.0174532925199433295]],
    PROJECTION["Krovak"],
    PARAMETER["False_Easting",0],
    PARAMETER["False_Northing",0],
    PARAMETER["Pseudo_Standard_Parallel_1",78.5],
    PARAMETER["Scale_Factor",0.9999],
    PARAMETER["Azimuth",30.28813975277778],
    PARAMETER["Longitude_Of_Center",24.83333333333333],
    PARAMETER["Latitude_Of_Center",49.5],
    PARAMETER["X_Scale",-1],
    PARAMETER["Y_Scale",1],
    PARAMETER["XY_Plane_Rotation",90],
    UNIT["Meter",1],
    AUTHORITY["EPSG","102067"]]
```

Výpis 6: Příklad WKT [15]

```
public void RefreshGpsOverlay()
{
  // Načítání geografických dat z paměti
  GpsOverlay.DataSource = new MemoryProvider(CreateGpsPoint(Map.Instance.GpsCoordinate));
}

public List<Mapsui.Geometries.Point> CreateGpsPoint(Mapsui.Geometries.Point sp)
{
  // Seznam geografických prvků k zobrazení
  var result = new List <Mapsui.Geometries.Point> ();
  result.Add(sp);
  return result;
}
```

Výpis 7: Přidání geografických prvků do mapové vrstvy

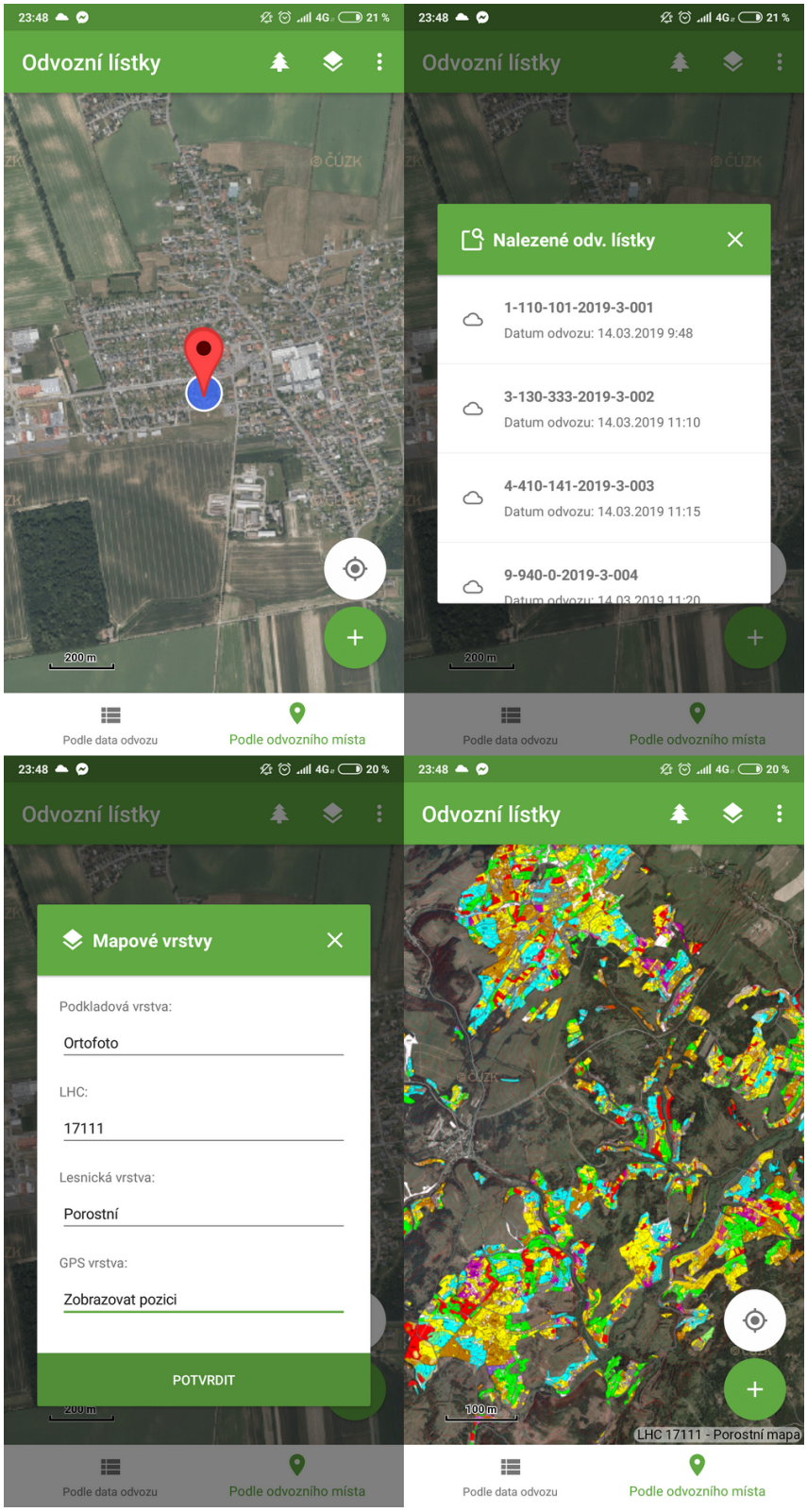
5.1.10.5 Zobrazení odvozních lístků v mapě

Jako optimální způsob zobrazení odvozních lístků v mapě se mi zdály piny - takové, jaké známe třeba z *Google Maps*. Knihovna *Mapsui.Forms* s nimi přímo počítá, což značně usnadnilo práci. Celý úkol spočíval v načtení seznamu existujících odvozních lístků z databáze, transformaci uložených GPS souřadnic do *ESPG:102067* a vygenerování nového pinu pro každý z těchto lístků.

Pinům se dal přidat event handler pro kliknutí - čehož jsem se rozhodl využít a na kliknutí nabízet seznam odvozních lístků v okolí kliknutí - vzhledem k tomu, že může existovat více odvozů ze stejného místa.

Využil jsem znalostí, které jsem získal při práci s knihovnou *OpenLayers* při vývoji aplikace ProPlaMobile a rozhodl se použít knihovnu *RBush* pro vyhledávání v okolí kliknutí - jelikož by vyhledávání podle GPS pozice v lineárním seznamu bylo s rostoucím počtem odvozních lístků výrazně náročnější, zároveň mi RBush nabízí už implementované metody pro vyhledávání v rámci extentu (geografických hranic).

Při generování pinů tak zároveň přidám záznam do RBushe a po kliknutí v něm vyhledávám, zda se v okolí nenachází i další odvozní lístky. Nalezené odvozní lístky pak zobrazím uživateli. V rámci RBushe vyhledávám podle *WGS-84* souřadnic, jelikož je jednodušší provést transformaci z jakéhokoliv souřadnicového systému do *WGS-84*, než provádět transformace mezi různými systémy - náročnost implementace transformační třídy pak rychle roste.



Obrázek 5: Mapové okno v aplikaci PDS_OLi

5.1.11 Implementace exportu odvozního lístku do XLSX

Pro účely exportu odvozního lístku mi byl doporučen framework *Office OpenXML SDK*. Jako první jsem vyzkoušel, zda je balíček použitelný s Xamarin.Androidem a pokusil se vytvořit nový dokument v externím úložisti telefonu. Tento test byl úspěšný a tak jsem mohl pokročit k programování samotného exportu. Pro zjednodušení jsme použili šablonu - existující sešit aplikace Microsoft Excel, jelikož by generování takto komplexního souboru bylo zdlouhavé a složité. Cílem exportu tak bylo pouze doplnit příslušné kolonky v šabloně a uložit jako nový soubor.

```
using(var doc = SpreadsheetDocument.CreateFromTemplate(await GetExportSamplePath()))
{
  var sheet = (Sheet) doc.WorkbookPart.Workbook.Sheets.ChildElements[0];
  var worksheetPart = (WorksheetPart) doc.WorkbookPart.GetPartById(sheet.Id);
  var worksheet = worksheetPart.Worksheet;
  var rows = worksheet.GetFirstChild < SheetData > ().Elements < Row > ();
  foreach(var row in rows)
  {
    if (row.RowIndex == 1)
    {
      var cells = row.Elements < Cell > ();
      foreach(var cell in cells)
      {
        var cellName = GetColumnName(cell.CellReference);
        if (cellName == "E") cell.CellValue = new CellValue(listek.Odberatel.Popis);
      }
    }
  }
}
```

Výpis 8: Vyplnění buňky textem

Načítání URL adresy šablony probíhalo pomocí metody *GetExportSamplePath()*, která byla označená jako abstraktní a musí být implementována pro každou platformu zvlášť. Všechny členské metody exportní třídy se tak volají pomocí tzv. *Dependency Service*, která zajistí načtení spravné implementace pro užitou platformu. Tato služba je součástí knihovny *Xamarin.Forms*. Metoda *ExportService.ToXlsx()* pak vrací cestu k nově uloženému souboru, který je uživateli nabídnut ke sdílení - ať už pomocí elektronické pošty, nebo cloudových uložišť.

```
var res = await DependencyService.Get<ExportService>().ToXlsx(Ticket);
```

Výpis 9: Volání nativní metody pomocí Dependency Service

```
public bool SendXlsxEmail(string path)
{
  try
  {
    var file = new Java.IO.File(path);
    var pth = Android.Support.V4.Content.FileProvider.GetUriForFile(Forms.Context, "eu.pds.OLi.
        fileprovider", file);
    file.SetReadable(true, false);
    var email = new Intent(Android.Content.Intent.ActionSend);
    email.PutExtra(Intent.ExtraStream, pth);
    var mime = MimeTypeMap.Singleton;
    email.SetType(mime.GetMimeTypeFromExtension("xlsx"));
    Forms.Context.StartActivity(email);
    return true;
  }
  ...
}
```

Výpis 10: Sdílení XLSX souboru na Androidu

Na Androidu verze 7.0 a vyšší je pro sdílení souborů nutné použít tzv. *File Providery*. Každý takový provider přesně specifikuje, ze kterých cest je možné soubory poskytovat a každého providera je nutné registrovat v *Android manifestu*. File provider umožní udělení dočasných práv pro čtení a zápis - uživatel tedy pro sdílení nemusí udělit aplikaci trvalé právo pro čtení a zápis, jak to bylo nutné v minulosti [6].

5.1.12 Implementace přihlašování k ProPlaCloudu

I v tomto případě byla nutná nativní implementace - k projektu .NET Standard totiž nelze připojit odkaz na ASMX webovou službu. Po připojení tohoto odkazu k projektu Xamarin.Android se vygenerovaly proxy třídy, které byly ihned přístupné k užití. Poté stačilo volat příslušné metody tak, jak by byly běžnou součástí projektu. I přesto jsem se setkal s pádem aplikace při volání webových metod - řešením bylo neočekávat HTTP zprávy *100 - Continue*.

```
public bool TestSignIn(string username, string pass)
{
  if (username != null && pass != null)
  {
    using(var service = new TestService())
    {
      System.Net.ServicePointManager.Expect100Continue = false;
      service.Credentials = new NetworkCredential(username, Hash.GenerateSha256(pass));
      try
      {
        var response = service.AuthTest();
        if (response.Success) return true;
        else return false;
      }
      catch(WebException e)
      {
        return false;
      }
    }
  }
  else return false;
}
```

Výpis 11: Volání ASMX webové služby v Xamarin.Androidu

5.1.13 Implementace exportu odvozního lístku do ProPlaCloudu

Víceméně totožný úkol, jako ten předchozí - pouze komplikovanější v tom, že bylo nutné provést převod z jednoho datového modelu na druhý - zjednodušený pro co nejrychlejší přenos po síti. Dalším problémem bylo, že v SQLite databázi byly ukládány pouze odkazy na fotografie, nikoliv fotografie samotné. Před odesláním na cloud tak bylo nutné fotografie nejdříve načíst, poté zkomprimovat a teprve poté přiřadit jejich *Base64* reprezentaci k vytvořenému objektu odvozního lístku.

Poté už zbývalo pouze otestovat funkčnost nahrávání. Do budoucna plánuji nahrávat fotografie a zbytek odvozního lístku samostatně - úspěšné nahrání fotografií v lese s pomalým Edge připojením je totiž nereálné. Uživatel by tak odeslal odvozní lístek a fotografie by se nahrály hned poté, co by bylo k dispozici rychlejší připojení.

```
foreach(var foto in listek.Fotos)
{
  using(var stream = new MemoryStream())
  {
    BitmapFactory.DecodeFile(foto.Cesta).Compress(Bitmap.CompressFormat.Jpeg, 50, stream);
    var bytes = stream.ToArray();
    var str = Convert.ToBase64String(bytes);
    fotos.Add(new cz.propla.moje.OdvozniListekFoto
    {
      Id = new Guid(Hash.GenerateGuid()),
      Foto = str
    });
  }
}
```

Výpis 12: Načtení obrázku a jeho konverze do Base64

5.1.14 Vydání aplikace v beta kanálu Google Play

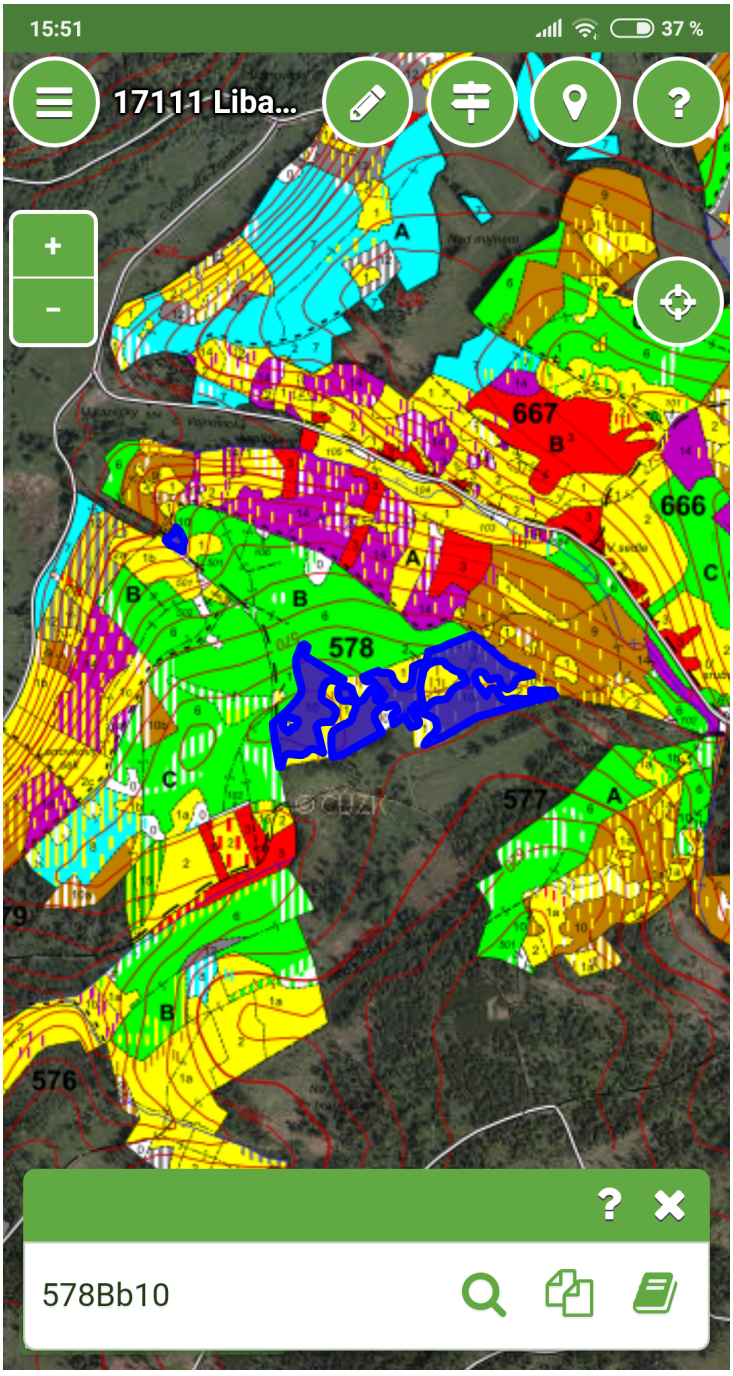
Na závěr bylo třeba aplikaci vydat v beta kanálu pro externí testery. Firma mi poskytla přístup k účtu na Google Play a pomocí webové *Google Play Console* jsem vytvořil novou stránku aplikace a nahrál sestavení. Následně jsem distribuoval odkaz na vstup do testovacího programu externím testerům.

5.2 Zjednoduššení tvorby uživatelských náčrtů v aplikaci ProPlaMobile

Aplikace ProPlaMobile nabízí svým uživatelům mapové okno, které mohou využít nejen pro prohlížení různých druhů map - od obyčejné ortofoto mapy až po speciální lesnické mapy, ale i pro kreslení vlastních náčrtů. Pro implementaci celé geografické složky aplikace je použita JavaScriptová knihovna *OpenLayers*. Tato knihovna už v základu umožňuje kreslení náčrtů do mapy. Pro potřeby aplikace ProPlaMobile byla tato funkcionalita lehce upravena - byla přidána překryvná vrstva přes upravovaný náčrt, která zvýrazňuje každý jednotlivý bod náčrtu a překryvná vrstva zobrazující délku úsečky mezi dvěma těmito body. Uživatelské náčrty lze kreslit jak ručně, tak i odečítáním aktuální GPS pozice uživatele - pohyb uživatele po reálném světě je pak zakreslen do mapy.

Problémem při ručním kreslení náčrtů na dotykovém displeji bylo, že uživatel svým prstem zakrýval upravovaný bod a nebyl tak schopen bod přesně pozicovat. Řešením tohoto problému byla implementace táhla pro posunování bodu. Táhlo je tlačítko na obrazovce posunuté o několik pixelů dolů pod upravovaný bod. Pohyb táhla mění pozici bodu - uživatel však prstem zakrývá táhlo, ne bod, a vidí tak přesně, na jaké pozici se upravovaný bod nachází.

Knihovna *OpenLayers* nabízí k samotnému kreslení i možnost přichytávání kresleného náčrtu k jiným vektorovým geografickým prvkům v mapě. Cílem druhé části tohoto úkolu bylo spojit implementaci táhla právě s touto funkcionalitou.



Obrázek 6: Mapové okno v aplikaci ProPlaMobile

5.2.1 Implementace táhla pro editaci uživatelských náčrtů

Před samotnou implementací táhla jsem se musel seznámit s kódem pro tvorbu náčrtů v aplikaci a se základním použitím knihovny OpenLayers. Pro implementaci táhla se v zásadě nabízely dvě možnosti - táhlo jako geografický prvek (featura) nebo táhlo jako DOM element.

Po diskuzi o kladech a záporech obou přístupů jsem se rozhodl pro implementaci táhla jako DOM elementu zobrazeného nad mapou. Pro pohyb (drag) táhla jsem použil interakci *Draggable* nabízenou knihovnou *jQuery UI*.

Bylo nutné, aby bylo táhlo provázané s upravovaným bodem. To znamenalo, že pohyb táhla měnil GPS pozici bodu a zároveň pohyb bodu měnil pozici táhla na obrazovce. Pro řešení obojího nabízí knihovna *OpenLayers* metody pro přepočty mezi pozicí v pixelech na obrazovce a GPS souřadnicemi.

```
1  /**
2   * Nastaví pozici táhla podle GPS souřadnic upravovaného bodu - při pohybu bodu se hýbe i táhlo
3   */
4  var _setCurrentPosition = function (map, coords) {
5    var pixel = map.getPixelFromCoordinate(coords);
6    if (pixel) {
7      pixel[0] -= 22;
8      pixel[1] += 50;
9      _pointMover.css({left: pixel[0], top: pixel[1]});
10   }
11 };
```

Výpis 13: Nastavení pozice táhla podle GPS souřadnic upravovaného bodu

```
1  /**
2   * Vrátí GPS souřadnice podle současné pozice táhla na obrazovce.
3   * Pokud je aktivní interakce Snap, ověřuje, zda je možné bod přichytit k jiné featuře.
4   * return {ol.Coordinates} coords GPS souřadnice bodu
5   */
6  var _getGPSCoordinates = function (map) {
7    var pixel = [parseFloat(_pointMover.css("left")) + 22, parseFloat(_pointMover.css("top")) -
         50];
8    var coords = map.getCoordinateFromPixel(pixel);
9    if (Snap.isActive()) {
10     coords = Snap.snapTo(pixel, coords);
11   }
12   return coords;
13 };
```

Výpis 14: Zjištění GPS pozice bodu podle pozice táhla na obrazovce

Protože jsem navázal táhlo na překryvnou vrstvu s body upravovaného náčrtu, bylo nutné po pohybu tímto bodem v překryvné vrstvě zrekonstruovat geografický prvek v mapě. K rekon-

strukci bodů a linií stačí obyčejné pole souřadnic bodů. Situace se komplikuje při rekonstruování polygonů a multipolygonů, které se v aplikaci rovněž používají.

Pro popis geografických prvků používá knihovna *OpenLayers* formát *GeoJSON*. Ten definuje polygon jako pole kruhů - ringů, které ho tvoří. Prvním kruhem je okraj polygonu, ostatní kruhy jsou případné díry uprostřed polygonu. Kruh se pak, stejně jako linie skládá z jednotlivých bodů. První a poslední bod kruhu musí být stejný. Multipolygon je pak definován jako pole polygonů.

Při rekonstrukci polygonů a multipolygonů tak nestačí pouze pole souřadnic bodů, ale je potřeba složitější struktura - nešlo by totiž určit, ke kterému kruhu body náleží. Pro odlišení mezi body/liniemi, polygony a multipolygony jsem tak k objektové definici náčrtu přidal dva příznaky - *poly* a *multi* určující, o jaký typ geometrie se jedná. Při samotné rekonstrukci tak aplikace nejprve zjistí, jakou geometrii má rekonstruovat a poté zavolá příslušnou metodu pro rekonstrukci.

```
1  /**
2   * Získá pole bodů v multipolygonu. Každý prvek v vráceném poli repzezentuje jeden polygon, každ
        ý polygon obsahuje své rings.
3   * Samotný ring pak obsahuje body, kterého ho tvoří (aby šlo táhlem editovat multipolygony s dí
        rami)
4   * Každý bod může být transformován pomocí mapFn
5   * @return {Array} ret Body multipolygonu.
6   **/
7  var _getMultiPolygonPoints = function (coords, mapFn) {
8    var ret = [];
9    if ($.isArray(coords) && coords.length > 0) {
10     coords.forEach(function (polygon, index) {
11       var poly = [];
12       polygon.forEach(function (ring, index) {
13         poly.push(_getFlatPoints(ring, true, mapFn));
14       });
15     ret.push(poly);
16     });
17   }
18   return ret;
19 };
```

Výpis 15: Tvorba hiearchie bodů, ze kterých lze zrekonstruovat multipolygon

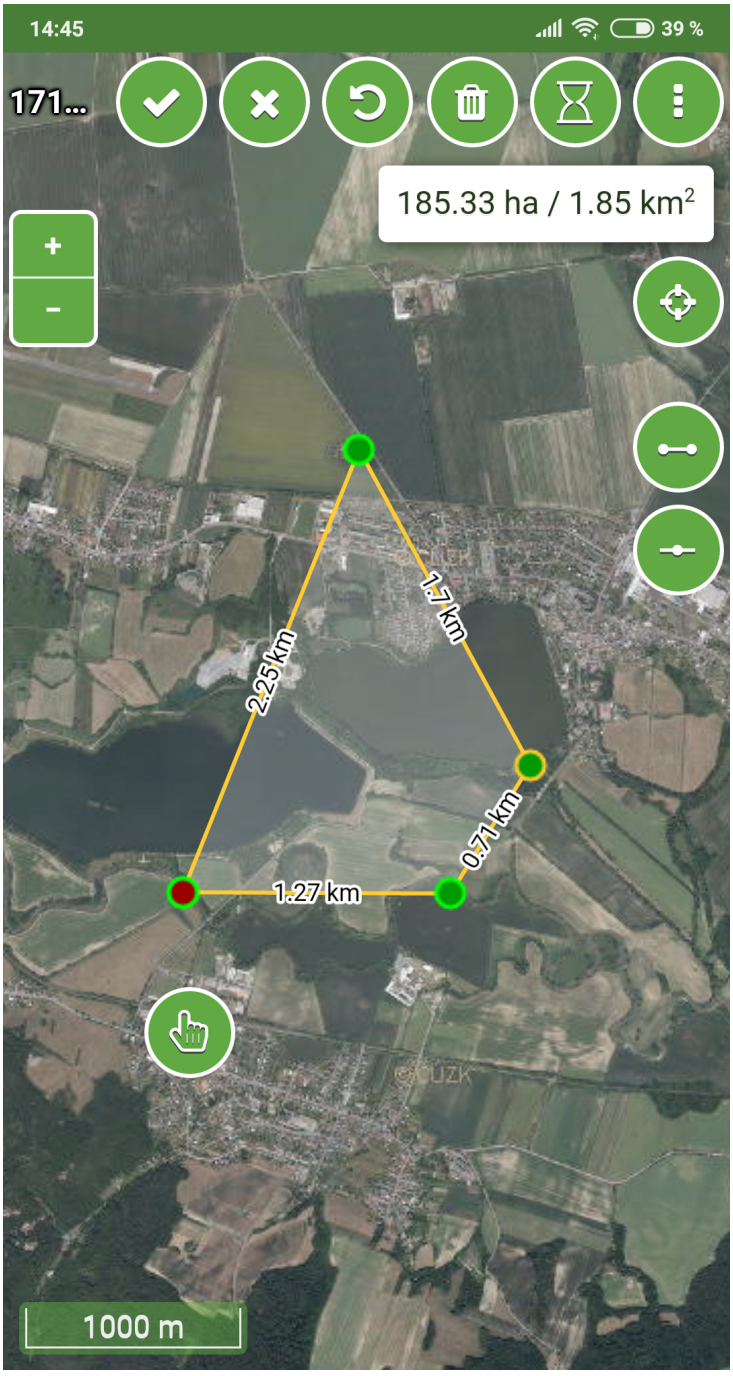
```
/**
 * Zrekonstruuje pole souřadnic ringu z pole bodů
 * @return {Array} ret Souřadnice featury
 **/
var _createRingCoords = function(points) {
    var ret = [];
    points.forEach(function(item, index) {
        ret.push(_copyCoords(item.getGeometry().getCoordinates()));
    });
    ret.push(ret[0]);
    return ret;
};
/**
 * Zrekonstruuje pole souřadnic polygonu z pole bodů
 * @return {Array} ret Souřadnice featury
 **/
var _createPolygonCoords = function(points) {
    var ret = [];
    points.forEach(function(item, index) {
        ret.push(_createRingCoords(item));
    });
    return ret;
};

/**
 * Zrekonstruuje pole souřadnic multipolygonu z pole bodů
 * @return {Array} ret Souřadnice featury
 **/
var _createMultiPolygonCoords = function(points) {
    var ret = [];
    points.forEach(function(item, index) {
        ret.push(_createPolygonCoords(item));
    });
    return ret;
};
```

Výpis 16: Rekonstrukce geografických prvků (featur) z bodů

Původní souřadnice geografického prvku pak jsou nahrazeny novými souřadnicemi zrekonstruovanými z hiearchie bodů v překryvné vrstvě. V závislosti na změně souřadnic jsou pak ještě přepočítány vzdálenosti mezi dvěma body zobrazované v druhé překryvné vrstvě, přidán nový záznam do *undo logu* a vypočítány indexy posledního bodu a změněného bodu, které mají odlišný barevný styl.

```
/**
 * Aktualizuje geometrii upravované featury na základě změny pozice bodu táhlem
 * @param record Upravovaný náčrt
 **/
var _refreshFeatureGeometry = function(record) {
    var geom = record.feature.getGeometry();
    if (geom instanceof ol.geom.LineString && record.points.length > 0) {
        geom.getGeometry().setCoordinates(_createLineStringCoords(record.points));
    }
    if (geom instanceof ol.geom.Polygon && record.points.length > 0) {
        geom.setCoordinates(_createPolygonCoords(record.points));
    }
    if (geom instanceof ol.geom.MultiPolygon && record.points.length > 0) {
        geom.setCoordinates(_createMultiPolygonCoords(record.points));
    }
};
```

Výpis 17: Nahrazení starých souřadnic prvků zrekonstruovanými souřadicemi



Obrázek 7: Vzhled mapového okna v aplikaci ProPlaMobile s implementovaným táhlem

5.2.2 Implementace přichytávání (snappingu) náčrtů za použití táhla

Dalším nápadem pro zjednoduššení ovládání tvorby náčrtů bylo přidání přichytávání k existujícím vektorovým prvkům v mapě. Tuto funkcionalitu nabízí knihovna *OpenLayers* v rámci interakce *Snap*. Mým úkolem tedy bylo tuto interakci přidat a zajistit, aby fungovala i při používání táhla.

```
var snap = new ol.interaction.Snap({
    features: new ol.Collection([]), // Ke kterým prvkům přichytávat
    edge: true, // Přichytávání k hranám
    vertex: true, // Přichytávání k vrcholům
    pixelTolerance: 15 // Tolerance přichytávání v pixelech
});

map.addInteraction(snap);
```

Výpis 18: Příklad inicializace interakce Snap

Takto přidaná interakce však funguje pouze při ruční úpravě bodu a na pohyb táhla nereaguje. Důvodem je, že interakce reaguje na událost *pointermove*, která se při pohybu táhlem nevyvolá. Zkoušel jsem tedy možnost vyvolat tuto událost ručně, ale nepodařilo se mi úspěšně zrekonstruovat argumenty nutné pro úspěšné provedení přichycení, které jinak událost *pointermove* posílá.

Rozhodl jsem se tedy prozkoumat přímo kód knihovny *OpenLayers* a případně jej upravit. Při průzkumu kódu interakce Snap jsem narazil na metodu *snapTo()*, která dostala jako argumenty pozici kliknutí na obrazovku v pixelech a souřadnice přichytávaného bodu. Na základě toho vracela buď nové souřadnice (tedy souřadnice bodu, ke kterému se upravovaný bod přichytil), nebo původní souřadnice. Ačkoliv není metoda v oficiální dokumentaci, není ani označená jako *@private*, což mi umožnilo ji použít a zprovoznit přichytávání pomocí táhla.

```
/**
 * Zkusí přichytit bod k některé z vektorových featur a vrátí buď staré,
 * nebo pozměněné souřadnice podle toho, zda bod lze,
 * nebo nelze přichytit (vzhledem k jeho souřadnicím a toleranci interace Snap)
 * @returns {ol.Coordinate} coords Souřadnice bodu
 */
var _snapTo = function(pixel, coords) {
    var snapResult = snapIntercation.snapTo(pixel, coords, snapIntercation.getMap());
    if (snapResult.snapped) {
        coords = snapResult.vertex.slice(0, 2);
    }
    return coords;
};
```

Výpis 19: Přichytávání pomocí táhla

Posledním problémem byla nutnost celou funkcionalitu optimalizovat. Počet prvků v mapě dosahuje díky načítání obsáhlých dat o lesních hospodářských celcích a zobrazování interaktivních lesnických map až několika tisíců. Načítání těchto dat, uložených ve formátu *GeoJSON* trvá několik sekund. Před přichytáváním je navíc nutné tyto data buď načíst znovu, nebo alespoň zkopírovat. V průběhu načítání dojde k zaseknutí aplikace - zpracování GeoJSONu neprobíhá asynchronně. Cílem tedy bylo dobu tohoto záseku minimalizovat.

Jako lepší varianta se jevilo prvky z vektorové vrstvy překopírovat, než znovu načítat. Při kopírování jsem použil *bulk insert* všech prvků z jedné vektorové vrstvy najednou. To nečekaně způsobilo zásek až na několik desítek sekund místo očekávaných jednotek sekund. Při průzkumů podobných témat na internetu jsem zjistil, že je z neznámého důvod *bulk insert* pomalejší, než vkládání po jednom prvku. Toto tvrzení se ukázalo jako pravdivé - délka záseku klesla z desítek sekund na zhruba jednu až dvě sekundy, což je při vhodném zobrazení informační obrazovky takřka nepostřehnutelné.

I tak jsem se dále snažil, aby bylo nutné prvky ideálně načítat pouze jednou, ne pokaždé, jak to bylo doposud. Zvolil jsem tedy variantu jednoho vygenerování interakce *Snap* a její další znovupoužití pouze změnou vlastností. Podle dokumentace sice interakce nabízí možnost vlastnosti měnit, jejich změna už pak ale není implementovaná, což znamená, že je nutné interakci generovat znovu. Rozhodl jsem se rozšířit knihovnu o dvě nové metody pro nastavení vlastností, které jsem potřeboval měnit.

```
1  /**
2   * Rozšíření knihovny OpenLayers
3   * Nastaví privátní parametr edge_.
4   * @param edge
5   */
6  ol.interaction.Snap.prototype.setEdge = function(edge) {
7      this.edge_ = edge;
8  };
9
10 /**
11  * Rozšíření knihovny OpenLayers
12  * Nastaví privátní parametr vertex_.
13  * @param vertex
14  */
15 ol.interaction.Snap.prototype.setVertex = function(vertex) {
16     this.vertex_ = vertex;
17 };
```

Výpis 20: Rozšíření knihovny OpenLayers - změna vlastností

Takto optimalizovaná interakce již nezpůsobuje zpomalování aplikace - vše se načte pouze jednou a poté se mění vlastnosti interakce za běhu. Uživateli je nabídnuto přichytávat pouze k hranám, pouze k vrcholům nebo obojí dohromady. Zároveň si může v nastavení aplikace sám nastavit toleranci interakce - při použití táhla stačí malá tolerance, pokud však posunuje přímo body ručně nebo stylusem, je vyšší tolerance výhodou.

5.3 Rozšíření webové služby ProPlaCloud

Aplikace ProPlaMobile umožňuje import a export dat z/do aplikace PDS_ProPla pomocí jednoduché webové služby ProPlaCloud. Cílem tohoto úkolu bylo umožnit stejný export i z aplikace PDS_OLi do aplikace PDS_ProPla.

5.3.1 Implementace webové služby pro nahrávání a stahování odvozních lístků z aplikace PDS_OLi

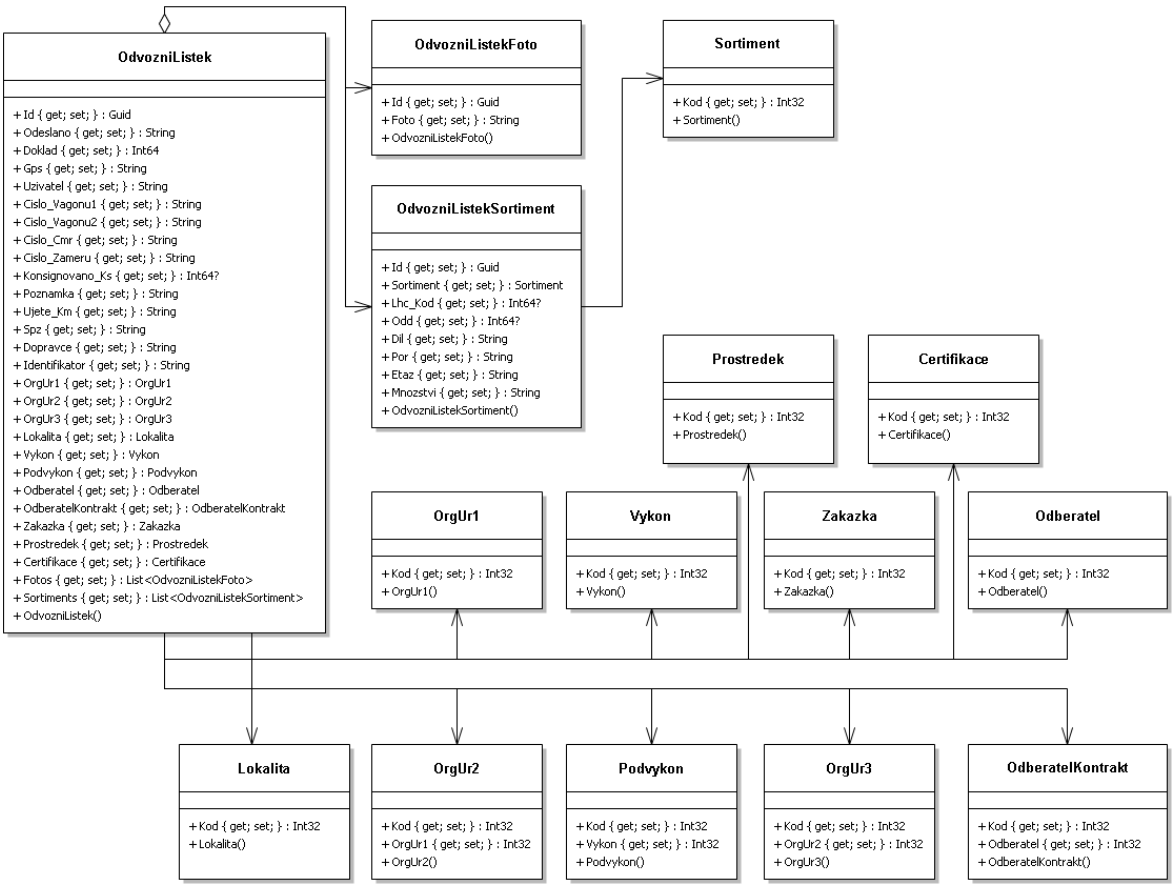
Po základním seznámení se s kódem jsem mohl rovnou implementovat novou webovou službu pro odvozní lístky. Ta se kromě odlišných dat posílaných po síti neliší od ostatních služeb, co ProPlaCloud nabízí.

Bylo důležité vymyslet takový datový model, který by minimalizoval množství dat posílaných po síti a zároveň dokázal předat vše, co bylo důležité pro převod odvozního lístku do aplikace PDS_ProPla. Jako potenciálně problematický se jevil přenos fotografií, které bylo před přenosem na cloud nutné zkomprimovat, zmenšit a zakódovat do *Base64*. Pro přenos dat číselníků (výkon, podvýkon, lokalita aj.) stačilo uvádět jednoznačný identifikátor každé položky - tedy kód, případně ještě kód nadřazené položky - např. u podvýkonu ještě kód nadřazeného výkonu.

Přijatý odvozní lístek je serializován do formátu *JSON* a uložen do složky podle přihlášeného uživatele a IMEI zařízení, ze kterého byl odvozní lístek odeslán. Poté je vrácena odpověď - pokud bylo uložení odvozního lístku úspěšné, vrací se v rámci odpovědi datum a čas uložení. Takto vytovřenou službu jsem pomocí IIS serveru otestoval na lokálním počítači a požádal jsem kolegu o nasazení na testovací server.

```
/// <summary>
/// Import odvozního lístku z OLi.
/// </summary>
/// <param name="IdZarizeni"></param>
/// <param name="Oli"></param>
/// <returns></returns>
[WebMethod]
public Json.OliResponse OliImport(string IdZarizeni, Model.Oli.OdvozniListek Oli) {
  try {
    string dirPath = _getUserDirPath();
    if (!Directory.Exists(dirPath)) {
      Directory.CreateDirectory(dirPath);
    }
    string newOlFile = Path.Combine(dirPath, IdZarizeni + "_" + Oli.Id + ".json");
    File.WriteAllText(newOlFile, JsonConvert.SerializeObject(Oli));
    return new Json.OliResponse() {
      Success = true,
      Message = "Odvozní lístek byl úspěšně uložen.",
      DatumPotvrzeni = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff")
    };
  }
  catch(Exception ex) {
    log.Error("Chyba ukládání odvozního lístku ze zařízení: " + IdZarizeni + ".", ex);
    return new Json.OliResponse() {
      Success = false,
      Message = "Chyba při ukládání odvozního lístku."
    };
  }
}
```

Výpis 21: Webová metoda pro nahrání odvozního lístku na cloud



Obrázek 8: Datový model odvozního lístku pro přenos na cloud

6 Znalosti a dovednosti uplatněné v průběhu odborné praxe

Při vývoji aplikace v Xamarinu jsem využil znalostí programovacího jazyka C#, které jsem nabyl v rámci předmětu *Programovací jazyky 2* a pokročilých znalostí technologie .NET, které mi poskytl předmět *Architektura technologie .NET.* V omezené míře jsem rovněž využil znalostí z předmětu *Tvorba aplikací pro mobilní zařízení II*, jelikož jsou dovednosti získané při programování Android aplikací v Javě použitelné i při programování v Xamarin.Androidu, který zachovává názvy metod i principy a pouze je přenáší do programovacího jazyka C#.

Dále jsem uplatnil znalosti a dovednosti získané v předmětech *Úvod do databázových systémů, Databázové a informační systémy* a *Vývoj informačních systémů* při návrhu a implementaci datového modelu a datové vrstvy aplikace. Neméně důležité byly i vědomosti, které mi poskytl předmět *Uživatelská rozhraní.*

Při vývoji aplikace v Apache Cordově a JavaScriptu jsem využil zejména znalostí, které jsem získal v předmětu *Tvorba aplikací pro mobilní zařízení* a v předmětu *Skriptovací programovací jazyky a jejich aplikace*, neboť se vývojem v JavaScriptu oba předměty více či méně zabývaly. Při implementaci webové služby jsem využil znalostí poskytnutých předmětem *Architektura technologie .NET*, který se implementací webových služeb okrajově zabýval.

7 Znalosti a dovednosti scházející v průběhu odborné praxe

V průběhu praxe jsem postrádal zejména znalosti z oblasti geoinformačních technologií a geodézie, které jsem si musel doplnit samostudiem - jedná se však o znalosti mimo rámec bakalářského studia informačních technologií. Nutné znalosti z oblasti lesnictví a lesního hospodářství mi pak studium informatiky nemohlo poskytnout už z principu - nejasnosti jsem však mohl kdykoliv konzultovat s kolegy, kteří mají lesnické vzdělání. Dále mi v průběhu praxe scházely znalosti (unit) testování aplikací, kterým se však později zabýval předmět *Architektura technologie .NET* - na praxi jsem se ale s unit testováním setkal dříve, než ve škole a musel jsem si proto chybějící znalosti dostudovat.

8 Závěr

V průběhu své odborné praxe jsem získal ucelený pohled na vývoj mobilních aplikací - od prvního návrhu až po vydání, seznámil jsem se s novými technologiemi a vyzkoušel si práci na skutečném projektu menšího rozsahu - výstupem je aplikace, která výrazně zjednodušší administrativu v oblasti těžby dřeva.

V mezičase jsem vylepšil existující aplikaci ProPlaMobile a jednu z jejích stěžejních částí - tvorbu uživatelských náčrtů. Ta je nyní díky použití táhla pro posun bodu dobře použitelná i na dotykových obrazovkách. Práce na existující aplikaci mi umožnila vyzkoušet si také práci s kódem, který napsal někdo jiný - ať už kolega, nebo vývojář třetí strany.

Odbornou praxi hodnotím velmi kladně. Díky samostatné práci na reálném projektu jsem výrazně prohloubil své znalosti z oblasti programování a návrhu software, což se projevilo kladně již v průběhu zimního semestru závěrečného ročníku na výrazném zlepšení studijních výsledků. Vzhledem k zaměření firmy jsem rovněž získal nové znalosti z oblasti lesnictví, geoinformatiky a geodézie, které chci v budoucnu dále prohlubovat. Práce na obou zmíněných projektech mě bavila a rád bych se nadále podílel na jejich vývoji a rozšiřování.

Literatura

[1] *PDS — Komplexní softwarová řešení* [online]. Brno, © 2007-2019 [cit. 2019-03-19]. Dostupné z: https://www.pds.eu/

[2] Android. In: *Techopedia* [online]. Techopedia, 2019 [cit. 2019-04-25]. Dostupné z: https://www.techopedia.com/definition/5415/android

[3] Android Debug Bridge (adb). *Android Developers* [online]. [cit. 2019-03-28]. Dostupné z: https://developer.android.com/studio/command-line/adb

[4] .APK File Extension. In: *FileInfo* [online]. Sharpened Productions, 2019 [cit. 2019-04-25]. Dostupné z: https://fileinfo.com/extension/apk

[5] Android SDK. *Techopedia* [online]. 2019 [cit. 2019-03-28]. Dostupné z: https://www.techopedia.com/definition/4220/android-sdk

[6] FileProvider. *Android Developers* [online]. [cit. 2019-04-23]. Dostupné z: https://developer.android.com/reference/android/support/v4/content/FileProvider

[7] ART and Dalvik. In: *Android Open Source Project* [online]. [cit. 2019-04-25]. Dostupné z: https://source.android.com/devices/tech/dalvik

[8] Architectural overview of Cordova platform. In: *Apache Cordova* [online]. 2015 [cit. 2019-04-25].
Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

[9] About SQLite. In: *SQLite* [online]. [cit. 2019-04-25]. Dostupné z: https://www.sqlite.org/about.html

[10] Web Map Service. In: *OGC* [online]. 2019 [cit. 2019-04-25]. Dostupné z: https://www.opengeospatial.org/standards/wms

[11] MBTiles. *OpenStreetMap Wiki* [online]. OpenStreetMap Wiki, 2017 [cit. 2019-03-28]. Dostupné z: https://wiki.openstreetmap.org/wiki/MBTiles

[12] Tour of .NET. In: *Microsoft Docs* [online]. 2017 [cit. 2019-04-25]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/standard/tour

[13] .NET Standard. In: *Microsoft Docs* [online]. 2017 [cit. 2019-04-25]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/standard/net-standard

[14] EXtensible Application Markup Language (XAML). In: *Microsoft Docs* [online]. 2019 [cit. 2019-04-25].
Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/xaml/

[15] ESRI:102067. *Epsg.io* [online]. 2018 [cit. 2019-03-28]. Dostupné z: https://epsg.io/102067